

Zian Wermelinger | M326 | 1 April 2024

Stock Market Exchange Simulator

Java Project

Inhalt

[Projektbeschrieb 2](#_Toc510771281)

[Wie starte ich das Projekt? 3](#_Toc510771282)

[Anforderungen 3](#_Toc510771283)

[Nicht funktionale Anforderungen 3](#_Toc510771284)

[Funktionale Anforderungen 3](#_Toc510771285)

[Use-Case-Modell 3](#_Toc510771286)

[UML des Klassensystems 3](#_Toc510771287)

[Kontextdiagram 3](#_Toc510771288)

[Verwendete Libraries 3](#_Toc510771289)

[Verwendete Konstanten 4](#_Toc510771290)

[Schnittstellenbeschreibung 4](#_Toc510771291)

[Welche Fehlerzustände können auftreten 4](#_Toc510771292)

[Wo werden Fehler geloggt 4](#_Toc510771293)

[Testszenarien erstellt und dokumentiert 4](#_Toc510771294)

[Implementierungsentscheidungen 4](#_Toc510771295)

[In Bezug auf abstrakte Klassen 4](#_Toc510771296)

[In Bezug auf Klassenhierarchien 4](#_Toc510771297)

[In Bezug auf Interfaces 4](#_Toc510771298)

[Zugriff auf das Datenbanksystem 4](#_Toc510771299)

[Datensicherheit & Applikationssicherheit 4](#_Toc510771300)

# Projektbeschrieb

Das Projekt hat den Namen „*Stock Market Exchange Simulator*“ oder abgekürzt auch **SMES**. Das zuvor erwähnte Produkt soll keineswegs einen wahren „*Simulator*“ nachstellen, sondern es soll die verschiedenen Möglichkeiten eines veränderbaren Kurses, welches sich in einer fiktionalen Umgebung befindet und sich in einem begrenzten Zeitraum abspielt, darstellen.

**SMES** sollte als eine **Java**-Applikation realisiert werden und zwar durch **JavaFX**. Es werden verschiedene Design Patterns verwendet, um die Realisierung effizient zu ermöglichen. In der folgenden Liste werden die Patterns aufgezählt.

* Observer Pattern
* Singleton
* Factory

Diese werden für diverse Aktivitäten benötigt, um das Grundgerüst der Applikation am Laufen zu halten. Die Integrität des gesamten Programmes wäre ohne diese Muster nicht im Bereich des Möglichen, weshalb dies der Grund ist, dass diese hier verwendet werden müssen.

In diesem Absatz werden kurz alle Features erwähnt, welche bis dato verwendbar sind.

* Login
* Registrierung
* Create Stock
* View Stock Information
* Update Stock Information
* Show Stock changes

Dies sind die Grundfunktionen, welche benötigt werden um das Minimum von **SMES** zu realisieren.

**SMES** ist ein Projekt, welches semi-schwer ist um die Realisierung umzusetzen. Jedoch wollte ich es so real wie möglich machen, deshalb habe ich auch viele Formeln benötigt, die zur Berechnung des Kurses und zur Vorhersagung der Werte benötigt werden, welche auf der Wahrscheinlichkeitstheorie basieren.   
Unter anderem werden die „*Las Vegas“* und die *„Monte Carlo“* Funktionen eingesetzt, zusätzlich dazu kommt eine kleine selbst erstellte Funktion zum Einsatz, welche die Veränderung des Wertes effizient im Zusammenhang mit Angebot-Nachfrage arbeitet. Ich nenne sie den „*Demand-And-Supply-Koeffizient*“.

# Wie starte ich das Projekt?

Um das Projekt zu starten werden verschiedene Voraussetzungen benötigt:

* Eine funktionierende Java-IDE die auch JavaFX unterstützt
* Genug Speicher um das Projekt auf der Festplatte zu laden
* Standard Hardware, wie Maus, Tastatur etc
* Installierte Treiber für Java

Sobald diese Bedingungen vorausgesetzt und bestätigt wurden, so steht dem Start der Applikation nichts mehr im Wege.   
Dafür müssen die folgenden Schritte befolgt werden um das Projekt starten zu können.

1. Starten der Java-IDE
2. Bei Anfrage, ob man ein bestehendes Projekt öffnen möchte, diese bestätigen
   1. Und den Pfad und Ordner angeben
   2. Ansonsten unter “File -> Open Project“
3. Sobald das Projekt vollständig geladen hat
   1. Auf „Run“ klicken
   2. Oder Tastenkombination verwendet (IntelliJ -> Shift+F10)

# Anforderungen

In diesem Kapitel werden alle Anforderungen aufgelistet, welche für das Projekt definiert wurden. Einige waren vorgegeben, andere wiederum wurden selbst beschrieben.

## Nicht funktionale Anforderungen

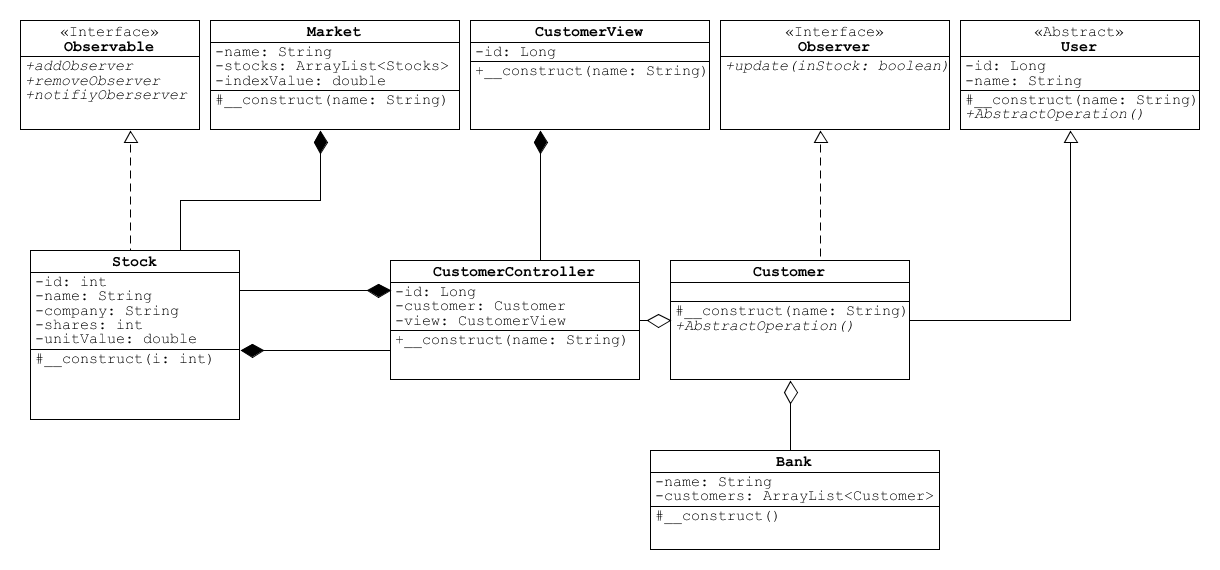
* Fehlerlogging mittels Output-Stream korrekt umgesetzt
* Korrekte Implementierung &Verwendung des Erzeugungsmusters: Singleton
* Korrekte Implementierung & Verwendung des Erzeugungsmusters: Factory
* Korrekte Implementierung & Verwendung des Verhaltensmusters: Observer
* Korrekte Implementierung des Architekturmusters: Model-View- Controller / oder MVVM
* Verwendung von mindestens 3 selbst erstellten Interfaces korrekt und sinngemäß für
* das Projekt
* Verwendung von mindestens 3 abstrakten Klassen korrekt und sinngemäß für das Projekt
* Verwendung von mindestens 3 Basisklassen korrekt und sinngemäß für das Projekt
* Abstrahierung der Datenbankschnittstelle auf einzelne Entitäten korrekt und
* sinngemäß umgesetzt
* Programm ist lauffähig und stürzt nicht ab
* Testszenarien gemäß Dokumentation implementiert und sind Funktionsbereit
* Umsetzung mindesten zweier Entitäten mittels CRUD-Operationen
* GUI ansprechend und sinngemäß gestaltet und umgesetzt

## Funktionale Anforderungen

* c
* c
* c
* c
* c
* c
* c

# Use-Case-Modell

# UML des Klassensystems



# Kontextdiagram

# Verwendete Libraries

* JDBC
  + Zur Verwendung von Datenbanken
* MySQL-Connector
  + Um mit der MySQL Datenbank verbinden zu können.
* JUnit
  + Um die Unit Tests ausführen und realisieren zu können

# Verwendete Konstanten

# Schnittstellenbeschreibung

# Welche Fehlerzustände können auftreten

# Wo werden Fehler geloggt

# Testszenarien erstellt und dokumentiert

# Implementierungsentscheidungen

## In Bezug auf abstrakte Klassen

## In Bezug auf Klassenhierarchien

## In Bezug auf Interfaces

# Zugriff auf das Datenbanksystem

# Datensicherheit & Applikationssicherheit